INF05516 - Semântica formal N
Ciência da Computação - UFRGS
2006-2

Marcus Ritt
mrpritt@inf.ufrgs.br

06/09/2006

Agenda

Última aula:

- Cálculo lambda.

Hoje:

- Linguagens realísticas: Extensões de IMP.

Extensões possíveis

Enriquecemos nossa linguagem com

- Paralelismo
- Laços diferentes (`for`)
- Expressões condicionais
- Efeitos laterias (`l++`)

O que tem mais?

- Declarações e tipos?
- Funções e procedimentos?
- Tratamento de exeções?
- Apontadores?
- Saltos locais: break e continue?
- Saltos global: goto ?!
- Entrada/saida?

Tratar erros na execução

- Nosso definição do IMP não contém a divisão de números intereiros.
- Podemos redefinir as expressões aritméticas

$$\circ_A = \{+, -, *, /\}$$
$$a ::= \mathbf{n} | \mathbf{x} | \mathbf{a_1} \circ_A \mathbf{a_2}$$

- e as regras

$$\frac{}{\mathtt{n}, \sigma \Downarrow n}\ \text{num} \qquad \frac{}{\mathtt{x}, \sigma \Downarrow \sigma(x)}\ \text{ident}$$

$$\frac{\mathbf{a_1}, \sigma \Downarrow n_1 \qquad \mathbf{a_2}, \sigma \Downarrow n_2}{\mathbf{a_1} \circ_\mathbf{A} \mathbf{a_2}, \sigma \Downarrow n}\ \text{aop, com } n = n_1 \circ_A n_2$$

Divisão por zero

- Infelizmente, as regras não funcionam no caso da expressão `1/0`: O valor não é definido!
- Para resolver esse problema temos diferenciar entre a divisão por 0 ou algum outro valor.
- Nossos opções contém os casos

  1. Não definir uma regra.
  2. Produzir um erro.

  Conseqûencias?

Semântica de erros

- Para tratar erros, nos vamos aumentar as categorias semânticas com um *valor de erro*:
  - Uma expressão aritmética tem um valor em $\mathbb{Z} \cup \mathrm{ERRO}$
  - Uma expressão booleana tem um valor em $\mathbb{B} \cup \mathrm{ERRO}$
  - Um comando tem um valor em $\Sigma = [\mathrm{Ident} \to \mathbb{Z}] \cup \mathrm{ERRO}$

Divisão com erros

$$\frac{\mathbf{a_1}, \sigma \Downarrow n_1 \qquad \mathbf{a_2}, \sigma \Downarrow n_2}{\mathbf{a_1}/\mathbf{a_2}, \sigma \Downarrow n} \text{ div, com } n_2 \neq 0, n = n_1/n_2$$

$$\frac{\mathbf{a_1}, \sigma \Downarrow n_1 \qquad \mathbf{a_2}, \sigma \Downarrow n_2}{\mathbf{a_1}/\mathbf{a_2}, \sigma \Downarrow \mathrm{ERRO}} \text{ div, com } n_2 = 0$$

Propagação de erros

$$\frac{\mathbf{a_i}, \sigma \Downarrow \mathrm{ERRO}}{\mathbf{a_1} \circ_{\mathbf{A}} \mathbf{a_2}, \sigma \Downarrow \mathrm{ERRO}} \text{ aop, com } \circ_A \in \{+, -, *, /\}$$

- Temos que definir regras para propagação de erros para todos expressões booleanas e comandos.
- Em geral: Com regras dirigidas pela sintaxe, se uma componente tem o valor ERRO a expressão composta também.
- Convenção: Erros são propagados implicitamente; não vamos escrever as regras.

Estado parcial 11 / 27

Motivação

- Para facilitar a definição de IMP, usavamos estados totais

$$\Sigma = [\mathrm{Ident} \to \mathbb{Z}]$$

- Uma alternativa é uma modelagem com estados parciais:

$$\Sigma = [\mathrm{Ident} \rightharpoonup \mathbb{Z}]$$

- Notação extendida:

$$\sigma[x \mapsto \mathrm{undef}](y) = \begin{cases} \sigma(y) & \text{se } x \neq y \\ \text{não definido} & \text{caso contrário} \end{cases}$$

Valores "default"

- Com estado total, qual é o estado inicial na execução? Isso vai ser relevante para varíaveis locais também?
- Algumas linguagens (C, C++) não definem o valor inicial; na prática, o valor inicial é o valor que se encontra casualmente na memôria.
- Valores "default" também tem desvantagens:
  - Eles não escalam: Qual é um valor default adequada para "expressões", "funções", etc.?
  - Menor desempenho, se o programador precisa um outro valor inicial.

Regras com estados parciais

Temos que modificar as regras em relação com variáveis:

$$\frac{x \in \mathrm{dom}(\sigma)}{\mathtt{x}, \sigma \Downarrow \sigma(x)}\ \mathrm{ident}$$

$$\frac{x \notin \mathrm{dom}(\sigma)}{\mathtt{x}, \sigma \Downarrow \mathrm{ERRO}}\ \mathrm{ident}$$

Consequência: Ler uma variável sem atribuir um valor antes resulta em um erro.

Introdução

Considere o seguinte exemplo em C

```
int x = 0;

int f() {
  return x;
}

int main(int argc, char *argv[]) {
  int x = 1;
  int v = f();
  // o que é o valor de v?
}
```


Introdução (2)

Compare com o seguinte exemplo de Perl

```
$x = 0;

sub f {
  return $x;
}

sub main {
  local $x = 1;
  $v = f();
  // qual é o valor de $v
}
```

Escopos

- Escopos contém declarações locais.
- Uma strategia parecida com C, se chama *escopo stático*
- Uma strategia parecida com Perl, se chama *escopo dinámico*
- Os escopos podem acontecer isolados ou no contexto de procedimentos ou funções.

Escopos simples

- Quais possibilidades temos de definir escopos?
- Escopos com declarações e com ou sem valores inicias?

```
begin
  var x;
  var y :=3;
  c;
end
```

- Escopos parecido com OCaml

```
let x := 3 in c
```

Escopos com let

$$c ::= \cdots \mid \texttt{let var x:=a in c} \mid \cdots$$

Exemplo:

```
let var x:=2+3 in
  let var y:=x*2 in (
    y:=y+1;
  )
```

Regra para `let`:

$$\frac{\mathbf{a}, \sigma \Downarrow n \qquad \mathbf{c}, \sigma[x \mapsto n] \Downarrow \sigma'}{\texttt{let var x:=a in c}, \sigma \Downarrow \sigma'[x \mapsto \sigma(x)]}\ \text{let}$$

Açucar sintático

- Com a definição de `let` podemos definir `begin ... end`
- Um escopo

  $$\texttt{begin}\ (\texttt{var x:=a})^*\ \mathbf{c}\ \texttt{end}$$

  corresponde com

  $$(\texttt{let x:=a in})^*\ \mathbf{c}$$
- Esse tipo de tradução se chama *açucar sintático* (inglês: syntactic sugar)
- Em geral açucar sintático é uma extensão da sintaxe que
  - tem uma tradução para expressões definidas (o núcleo da linguagem);
  - não aumenta a expressividade mas que
  - faz a programação mais "doce".

Declarações obrigatórias

- Com `let` a declaração de variáveis não é obrigatório.
- Uma possibilidade de garantir declarações: Proibir a atribuição a variáveis que não são declarados:

$$\frac{\mathbf{a}, \sigma \Downarrow n \qquad x \in \mathrm{dom}(\sigma)}{\mathbf{x}\texttt{:=}\mathbf{a}, \sigma \Downarrow \sigma[x \mapsto n]}\ \text{assign}$$

$$\frac{\mathbf{a}, \sigma \Downarrow n \qquad x \notin \mathrm{dom}(\sigma)}{\mathbf{x}\texttt{:=}\mathbf{a}, \sigma \Downarrow \mathrm{ERRO}}\ \text{assign}$$

- Desvantagem: Uma variável não declarado pode ser detectado estáticamente, mas nossa solução produz um erro no tempo da execução.

Análise estática

- Idéia básica: Um programa é bem formado, se cada variável ocorre num escopo de um `let` com o nome dessa variável.
- Em outras palavras: Um `let` funciona como "binder"; uma variável é *ligada* no escopo de um let, ou *livre*.
- Logo, um programa é bem formado, se a conjunto de variáveis livres é vazio. (Compare com o cálculo lambda).

$$L(\texttt{skip}) = \emptyset$$
$$L(\mathbf{x}\texttt{:=}\mathbf{a}) = L(\mathbf{a}) \cup \{x\}$$
$$L(\mathbf{c_1}\texttt{;}\mathbf{c_2}) = L(\mathbf{c_1}) \cup L(\mathbf{c_2})$$
$$L(\texttt{if}\ \mathbf{b}\ \texttt{then}\ \mathbf{c_1}\ \texttt{else}\ \mathbf{c_2}) = L(\mathbf{b}) \cup L(\mathbf{c_1}) \cup L(\mathbf{c_2})$$
$$L(\texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c}) = L(\mathbf{b}) \cup L(\mathbf{c})$$
$$L(\texttt{let}\ \mathbf{x}\texttt{:=}\mathbf{a}\ \texttt{in}\ \mathbf{c}) = L(\mathbf{a}) \cup (L(\mathbf{c}) \setminus \{x\})$$

Sistema de tipos

- Uma outra possibilidade é de definir um sistema de tipos para IMP.
- Um sistema de tipos consiste em
  - Um conjunto de tipos $\tau \in T$
  - Um ambiente de tipos $\gamma \in \Gamma$
  - Regras que permitem deducir uma relação $\vdash \subseteq \Gamma \times \Theta \times T$

    $$\Gamma \vdash_\Theta \mathbf{e} : \tau$$

    para qualquer categoria sintática Θ.
  - O *julgamento de tipos* consiste em aplicar as regras do sistema de tipos para decidir se uma expressão é bem tipada.

Exemplo: Sistema de tipos para escopos

- Idéia: Um programa é bem tipado se todos variáveis são declaradas.
- Como temos só números inteiros, o conjunto T não é importante.
- Uma relação $\vdash \Gamma \times \Theta$

  $$\gamma \vdash_\Theta \mathbf{c}$$

  com um ambiente de tipos $\gamma \in \Gamma = \mathcal{P}(\mathrm{Ident})$ é suficiente.

Tipos: Expressões

$$\frac{}{\gamma \vdash_A \mathbf{n}}\ \text{t-num} \qquad \frac{\mathbf{x} \in \gamma}{\gamma \vdash_A \mathbf{x}}\ \text{t-ident}$$

$$\frac{\gamma \vdash_A \mathbf{a_1} \quad \gamma \vdash_A \mathbf{a_2}}{\gamma \vdash_A \mathbf{a_1} \circ_\mathbf{A} \mathbf{a_2}}\ \text{t-aop}$$

$$\frac{}{\gamma \vdash_B \mathbf{t}}\ \text{t-bool}$$

$$\frac{\gamma \vdash_B \mathbf{b_1} \qquad \gamma \vdash_B \mathbf{b_2}}{\gamma \vdash_B \mathbf{b_1} \circ_\mathbf{B} \mathbf{b_2}}\ \text{t-bop}$$

$$\frac{\gamma \vdash_A \mathbf{a_1} \qquad \gamma \vdash_A \mathbf{a_2}}{\gamma \vdash_B \mathbf{a_1}\ \mathrm{cmp}\ \mathbf{a_2}}\ \text{t-bop}$$

$\circ_A = \{+, -, *, /\}, \circ_B = \{\wedge, \vee\}, \mathrm{cmp} = \{=, <, >, \leq, \geq\}$.

Tipos: Comandos

$$\frac{}{\gamma \vdash_C \texttt{skip}}\ \text{t-skip}$$

$$\frac{\gamma \vdash_A \mathbf{x} \qquad \gamma \vdash_A \mathbf{a}}{\gamma \vdash_C \mathbf{x}\texttt{:=}\mathbf{a}}\ \text{t-assign}$$

$$\frac{\gamma \vdash_C \mathbf{c_1} \qquad \gamma \vdash_C \mathbf{c_2}}{\gamma \vdash_C \mathbf{c_1}\texttt{;}\mathbf{c_2}}\ \text{t-seq}$$

$$\frac{\gamma \vdash_B \mathbf{b} \qquad \gamma \vdash_C \mathbf{c_1} \qquad \gamma \vdash_C \mathbf{c_2}}{\gamma \vdash_C \texttt{if}\ \mathbf{b}\ \texttt{then}\ \mathbf{c_1}\ \texttt{else}\ \mathbf{c_2}}\ \text{t-if}$$

$$\frac{\gamma \vdash_B \mathbf{b} \qquad \gamma \vdash_C \mathbf{c}}{\gamma \vdash_C \texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c}}\ \text{t-while}$$

$$\frac{\gamma \vdash_A \mathbf{a} \qquad \gamma \cup \{x\} \vdash_C \mathbf{c}}{\gamma \vdash_C \texttt{let}\ \mathbf{x}\texttt{:=}\mathbf{a}\ \texttt{in}\ \mathbf{c}}\ \text{t-let}$$